REMARKS/ARGUMENTS

Claims 1-22 are pending. No new matter is added. Reconsideration and allowance of the claims is respectfully requested.

I. 35 U.S.C. § 102, Anticipation

The Examiner rejects claims 1, 8-9, and 16 under 35 U.S.C. § 102 as being anticipated by *Sawdon et al.,* Writable File System Snapshot with Ditto Address Feature (U.S. 7,085,785) (hereinafter "*Sawdon*"). This rejection is respectfully traversed.

The Examiner states:

> *Sawdon* discloses a method in a data processing system for storing data in a file system ("data storage systems", col. 1, lines 19-22), the method comprising determining whether space isavailable in an inode for a file in the file system ("determining if an inode to be modified in the specified snapshot is an empty inode", claim 1, 9, 17); and responsive to space being available, storing the data in the inode ("response to determining the inode to be modified is an empty inode", claim 1, 9, 17).

Office Action of December 27, 2007, page 2.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case, each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Claim 1 is a representative claim of this grouping of claims. Applicants first address the rejection of claim 1. Claim 1 is as follows:

> 1. A method in a data processing system for storing data in a file system, the method comprising:
>
> determining whether space is available in an inode for a file in the file system; and
>
> responsive to space being available, storing the data in the inode.

I.A. Rejection in view of *Sawdon*

Sawdon does not anticipate claim 1 because *Sawdon* does not teach all the features of claim 1. In rejecting claim 1 the Examiner cites to the following portion of *Sawdon:*

> This invention relates to data storage systems and more specifically to data storage systems that store snapshots (i.e., indications of the status of stored data at particular points in time).

Sawdon, col. 1, lines 19-22

Sawdon teaches a method and system for manipulating a specific type of inode, i.e., those associated with **snapshots**. The snapshot inode is completely different from the inode associated with a file within a file system. This snapshot inode contains status information regarding the state of a file just prior to the file being modified, or at the time the snapshot was captured. The snapshot inode only exists when a file on a file system is being modified. *Sawdon* col. 8, lines 9-16 states:

> The exemplary embodiments of the present invention capture one or more snapshots of a file system to create a data set that preserves the state of the data that was stored within that file system at the time the snapshot was captured. It is desirable to create and capture snapshots that include all files in a file system in order to maintain a consistent file system image and efficiently copy the old data in the file system prior to modification after capturing the snapshot.

The Examiner also cites to two features claimed in claims 1, 9, and 17 of *Sawdon*: determining if an inode to be modified in the specified snapshot is an empty inode and response to determining the inode to be modified is an empty inode. However, neither these features nor any other claimed features in *Sawdon* teaches "determining whether ***space*** is available ***in an inode for a file*** in the file system," as required by claim 1. Instead, *Sawdon* teaches that before data can be added to the inode, the inode must be ***empty***, i.e. no existing data can be in the inode. Moreover, this inode is one that is specified in the ***snapshot***, not the file. Arguably, if a file in *Sawdon* has not been modified, then no snapshot inode would exist for that file and therefore no processing of the inode would occur.

Additionally *Sawdon* fails to teach "responsive to space being available, storing the data in the inode", because the data in *Sawdon* is only stored when the inode is determined to be empty. Even assuming, *arguendo*, that a snapshot inode had space in it, *Sawdon* teaches that the data is still not stored in this inode, because the inode in this instance is not empty. Therefore, *Sawdon* fails to teach, "determining whether space is available in an inode for a file in the file system" and "responsive to space being available, storing the data in the inode." Accordingly, under the standards of *In re Bond*, *Sawdon* does not anticipate claim 1.

Claims 8, 9, and 16 contain features similar to those presented in claim 1. Hence, for the reasons presented above, *Sawdon* also does not anticipate these claims.

I.B. Rejection in view of *Crow*

The Examiner rejects claims 1-22 as anticipated by *Crow et al.*, Versatile Indirection in an Extent Based File System, U.S. Patent Application Publication 2004/0254907 (December 16, 2004) (hereinafter "*Crow*"). This rejection is respectfully traversed. Claim 1 is a representative claim of this grouping of claims. Claim 1 is as follows:

> 1. A method in a data processing system for storing data in a file system, the method comprising:
> determining whether space is available in an inode for a file in the file system; and
> responsive to space being available, storing the data in the inode.

The Examiner rejects claim 1 as anticipated by *Crow*. In regards to claim 1, the Examiner asserts that:

> Crow discloses in figure 8C, 9-11, a method in a data processing system for storing data in a file system, the method comprising determining whether space is available in an inode for a file in the file system (132, "determines whether at least one empty row remains for writing a new extent to the file's inode", paragraph [0041]); and responsive to space being available, storing the data in the inode (142, "If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode", paragraph [0042]).

Office Action of December 27, 2007, p 3.

Crow does not teach the features, "determining whether space is available in an inode for a file in the file system" and "responsive to space being available, storing the data in the inode" as claimed in claim 1. The Examiner asserts otherwise, citing *Crow*'s figure 8C, reproduced below:

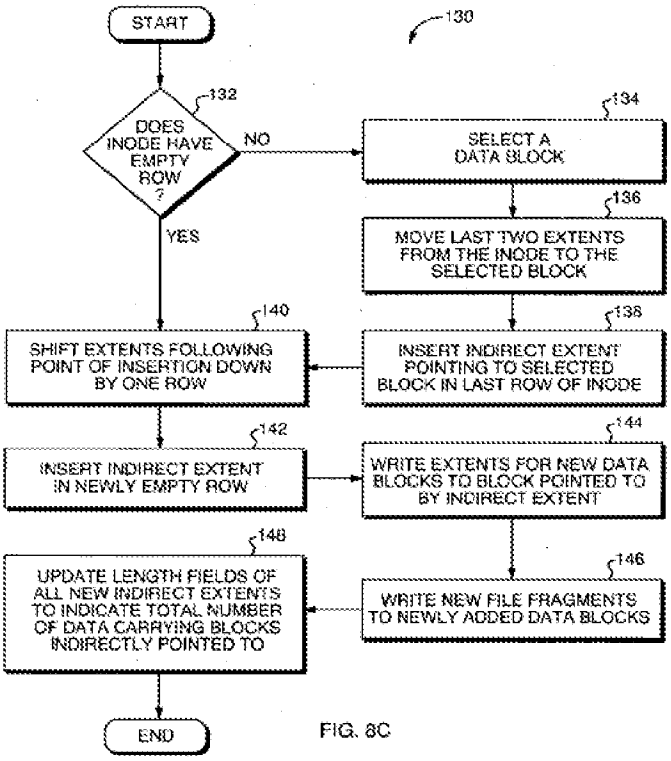

FIG. 8C

Crow describing Figure 8C states:

> [0041] FIG. 8C is a flow chart illustrating a method 130 of inserting a new file segment between two adjacent file segments. To insert the new segment, the operating system first determines whether at least one empty row remains for writing a new extent to the file's inode, for example to inode 110 of FIG. 8A (step 132). In FIG. 8A, the operating system would determine that the inode 110 does not have an empty row.
>
> [0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the operating system writes the new file segment to a new data block pointed to by the new direct extent (step 138).

Crow, pg3. paragraphs 0041 - 0042

According to figure 8C, in step 132, "the operating system first determines whether at least one empty row remains for writing a new *extent* to the file's inode." *Crow*, paragraph 41 (emphasis added). However, step 146 teaches that the new *file fragments are stored in newly added data block, not in the inode*. Moreover, *Crow*, paragraphs 0033-0034 states:

> [0033] Each extent of the illustrated embodiment has three fields including an address pointer field, a length field, and a flag field.
>
> [0034] The address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. In one embodiment, the pointer

fields for the logical volume and the data block therein are 2 bytes and 4 bytes long, respectively.

Crow, paragraphs 0033-0034

The above cited portion of C*row* teaches that the inode stores the extent, which contains a pointer that indicates both the logical volume and a physical offset of the data block. Therefore, figure 8C does not teach the features of claim 1 because figure 8C determines whether there is space in the inode for an "extent" and not for a file *in the file system* as claimed in claim 1. Furthermore, figure 8C discloses storing the data in a newly added data block instead of storing the data in the inode as claimed in claim 1. The Examiner also cites to Figures 9-11 as teaching the features of claim 1. Figure 9 illustrates:

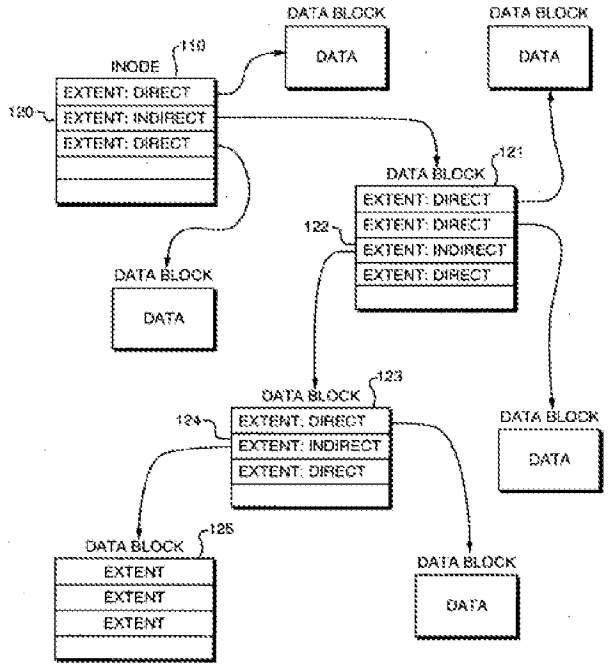

FIG. 9

Crow describing Figure 9 states:

[0045] FIG. 9 illustrates an example where the file system nests indirect extents. In the example, the inode 110 includes indirect extent 120, which points to data block 121. In turn, block 121 includes indirect extent 122, which points to data block 123, and block 123 includes indirect extent 124, which points to block 125.

[0046] Nesting indirect extents enables growing a file between any two original file segments without size limits. Nesting also introduces extra costs during accesses. Each access to a file segment pointed to by nested indirect extents costs extra look ups and additional look up time.

Crow, pg3. paragraphs 0045 - 0046

Figure 9 of *Crow* shows the relationship among inode 110 and a plurality of data blocks. Specifically, Figure 9 illustrates an example where the file system nests indirect extents. However,

nothing in Figure 9 teaches, "*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available...," as claimed in claim 1.

Figure 10 illustrates:

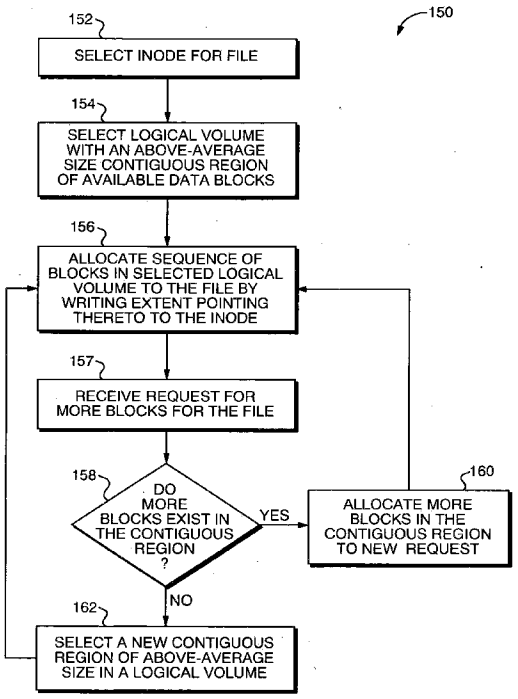

FIG. 10

Crow describing Figure 10 states:

> [0047] FIG. 10 is a flow chart illustrating a method 150 of allocating data blocks to a file from a plurality of logical volumes, for example, the volumes LV1, LV2 shown in FIG. 5. The operating system assigns an inode to the file by writing the inode address and the file name to a row in a directory (step 152). In FIG. 5, the operating system wrote the inode address for the inode 63, in entry of the root directory 61 for file name A. The operating system selects a logical volume with a larger than average contiguous region of available data blocks (step 154). The operating system determines the maximum number of available contiguous blocks in each logical volume from data in the volume's header or from information in a superblock spanning the entire storage system. The operating system allocates a string of data blocks from the contiguous region of the selected volume to the file by writing an extent, which points to the string, in the first row of the inode assigned to the file (step 156). The extent indicates both the logical volume and an offset of the first data block of the string of blocks within the selected logical volume.
>
> [0048] Later, a request from a software application for more data blocks for the file is received by the operating system (step 157). In response to the request, the operating system determines whether the region contiguous to the physical location of the previous segment of the file has more available data blocks (step 158). If region has more available blocks, the operating system allocates a new

> string of blocks immediately following the physical location previous segment, i.e., contiguous with the previous segment (step 160). Then, the operating system increases the value of the length stored in the length field of the previous extent for the region by the number of blocks in the new string (step 161). If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162). The newly found logical volume may be a different logical volume. Thus, the new string of data blocks may be allocated to the file from a different logical volume.

Crow, pg3. paragraphs 0045 - 0046

Figure 10 illustrates a method 150 of allocating data blocks to a file from a plurality of logical volumes. The operating system allocates a string of data blocks from the contiguous region of the selected volume to the file by writing an extent. The extent points to the string in the first row of the inode assigned to the file (step 156). Step 157 occurs when an application requests more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in figure 10 of *Crow* teaches the feature, "responsive to space being available, storing the data in the inode." Instead, figure 10 of *Crow* teaches a method of storing data in contiguous blocks, which is different from the features of claim 1. Therefore, figure 10 of *Crow* does not teach the features of claim 1.

Figure 11 illustrates:

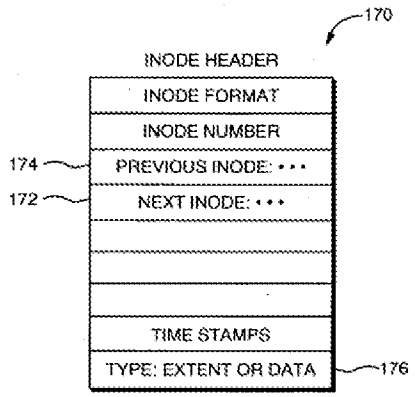

FIG. 11

Crow, describing Figure 11, states:

> [0050] FIG. 11 illustrates the headers 170 of one embodiment of the inodes 63, 64 of FIG. 5. The headers 170 provide the separate structures used to record the addresses of each inode. Each header 170 has entries 172, 174 for the addresses

> of the next inode to be allocated and of the previously allocated inode, respectively. These entries are written to the header 170 when the associated inode is first allocated.

Crow, pg 4. paragraphs 0050

Figure 11 of *Crow* shows details of an inode header and illustrates the headers 170 of one embodiment of the inodes 63, 64 of FIG. 5. Figure 11 of *Crow* shows that an inode reader includes format, number, previous inode, next inode, time stamps, and the type - whether extent or data. However, nothing in Figure 1 teaches, "*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available...," as claimed in claim 1.

As shown above, nothing in Figures 9-11, or any of the cited disclosures, teaches the claimed features claim 1. Therefore, under the standards of *In re Bond*, *Crow* does not anticipate claim 1 or any other claim in this grouping of claims.

I.B.1. Claims 2, 3, 7, 10, 11, 15, 17, 18, and 22

The Examiner rejects claim 2, 3, 7, 10, 11, 15, 17, 18 and 22 as anticipated by *Crow*. Claim 2 is a representative claim of this grouping of claims. Regarding claim 2, the Examiner asserts in this office action:

> Crow discloses in figures 8A-8C and 10, to determining whether additional data being present; and responsive to the additional data being present, storing the additional data in a partially filled block of another file (paragraph [0038], [0039], [0042] and [0044]).

Office Action December 27, 2007, pg 3.

Crow does not anticipate claim 2 because *Crow* does not teach all the features of claim 2. Claim 2 is as follows:

> 2. The method of claim 1 further comprising:
> determining whether additional data is present; and
> responsive to the additional data being present, storing the additional data in a partially filled block of another file.

Because claim 2 depends from claim 1, the same distinctions between *Crow* and claim 1 apply to claim 2. Additionally, claim 2 claims other additional combinations of features not disclosed by the reference. Specifically, *Crow* does not teach the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. The Examiner asserts otherwise, citing the following figures:

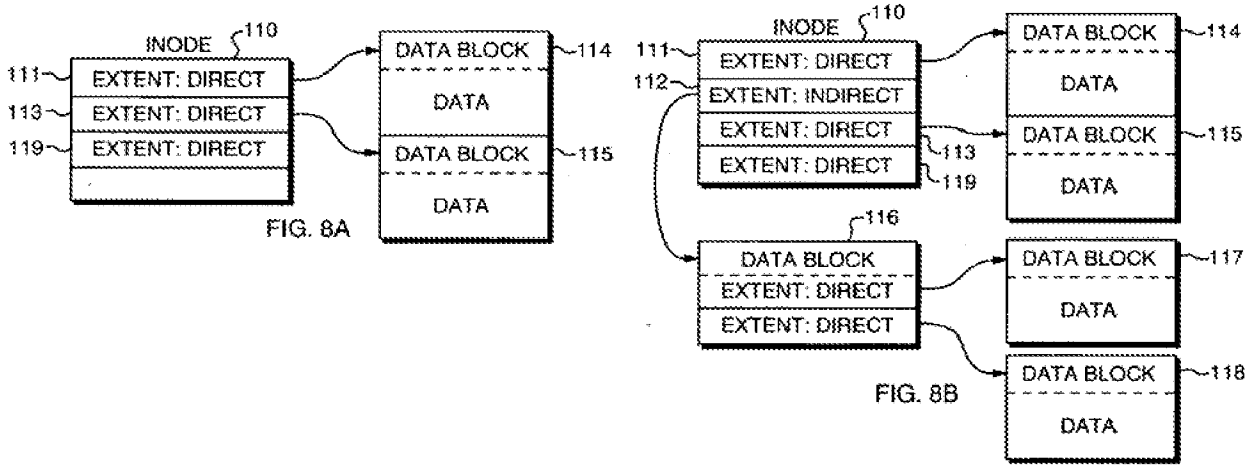

FIG. 8A

FIG. 8B

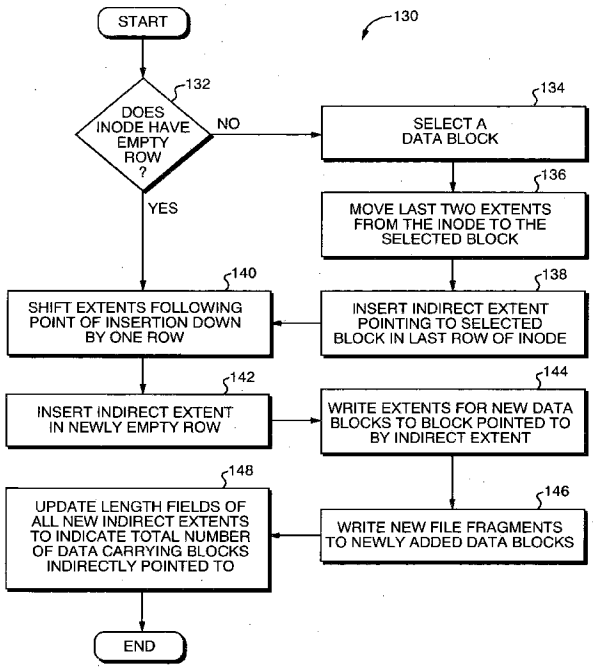

FIG. 8C

Figures 8A, 8B, and 8C do not teach the feature of, "storing *the additional* data in a *partially* filled block *of another file.*" Instead, figures 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to ***new*** data block 117 and original data block 215. *Crow*, paragraph 38. This feature enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. *Crow*, paragraph 39. Figure 8C is a flow chart illustrating a method 130 of inserting a new file segment between two adjacent file segments. *Crow*, paragraph 39. In step 148, the operating system writes the new file segment in the ***new*** data block pointed to by the new direct extent. *Crow*, paragraph 44.

However, these figures do not teach "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Instead, these figures show how to add ***new*** data blocks to a file using *indirect extents*. Therefore, figures 8A, 8B, and 8C of *Crow* do not teach the claimed features of claim 2.

Additionally, nothing in figure 10 of *Crow* teaches the feature, "storing *the additional* data in a *partially filled* block *of another file.*" Instead, figure 10 of *Crow* teaches a method of storing data in contiguous blocks, which is different from the features of claim 2. Figure 10 of *Crow* does not teach the features of claim 2. Moreover, as discussed above nothing in figure 10 of *Crow* teaches the features of claim 1. Claim 2 depends from independent claim 1. Therefore, at least by virtue of claim 2's dependence on claim 1, *Crow* does not anticipate claim 2.

Additionally, the Examiner cites to the following portions of *Crow* as disclosing the features of claim 2:

> [0038] FIGS. 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. FIG. 8A shows an inode 110 assigned to the file. The inode 110 has consecutive direct extents 111, 113, 119 that point to data blocks 114, 215, 330 storing originally consecutive segments of the file. FIG. 8B shows the final file in which an indirect extent 112 has been inserted between the two original direct extents 111, 119. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to **new** data block 117 and original data block 215. Since the indirect extent 112 is physically located between the two original extents 111, 119, the segments stored in the blocks 117, 215 (indirectly pointed to) are logically located between the original segments stored in the blocks 114, 330. Inserting the indirect extent 112 has grown the middle of the associated file by logically inserting the segment in **new** data block 117 between the originally consecutive segments in data blocks 114 and 215.

Crow, p. 3, ¶ 38 (emphasis added).

> [0039] The file system, illustrated in FIGS. 5-8B, allows any extent of an inode to be indirect, because the flag field indicates the type of each extent. This free placement of indirect extents within the inodes enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. To insert a new data segment, the system inserts an indirect extent into the file's inode between the two extents for the selected data segments. Then, the system makes the indirect extent point to a data block storing new direct extents that point, in turn, to the consecutive pieces of new data segment. The new direct extents are logically located in the inode at the point where the new indirect extent has been inserted.

Crow, p. 3, ¶ 39 (emphasis added).

> [0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the

> operating system writes the new file segment to a **new** data block pointed to by the new direct extent (step 138).

Crow, p. 3, ¶ 42 (emphasis added).

> [0044] Next, the operating system inserts an indirect extent into the row of the inode previously occupied by the extent now in the second row of the indirect block (step 146). The new indirect extent points to the new indirect block and has a length equal to the sum of the lengths of both extents in the indirect block. In FIG. 8B, the operating system writes the extent 112 pointing to the data block 116 to the inode 110. Finally, the operating system writes the new file segment in
> the **new** data block pointed to by the new direct extent (step 148). In FIG. 8B, the new file segment is written to the data block 117.

Crow, p. 3, ¶ 44 (emphasis added).

None of the portions cited by the Examiner or any other portion of *Crow* teaches the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Paragraphs 38 and 39 describe figures 8A and 8B. As previously discussed, this portion of *Crow* explains how to add ***new*** data blocks to a file using indirect extents. Similarly, paragraphs 42 and 44 describe figure 8C. Both paragraphs specifically state: "the operating system writes the new file segment in the ***new*** data block pointed to by the new direct extent." Therefore, none of the portions cited by the Examiner or any other portion of *Crow* teaches the feature of, "storing *the additional* data in a *partially filled* block *of another file,*" as recited in claim 2. Accordingly, *Crow* does not anticipate claim 2 or any other claim in this grouping of claims.

I.B.2. Claims 4, 12, and 19

Claim 4 is a representative claim of this grouping of claims. Claim 4 is as follows:

> 4. The method of claim 3, wherein the partially filled block is a last block of the another file.

The Examiner rejects claim 4 as anticipated by *Crow*. Regarding claim 4, the Examiner states that:

> Crow discloses in figure 8C, wherein the partially filled block being a last block of the another file (paragraph [0042]).

Office Action dated December 27, 2007, p. 4.

The portion of *Crow* cited by the Examiner is as follows:

> [0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the

> operating system writes the new file segment to a new data block pointed to by the new direct extent (step 138).

Crow, paragraph 0042.

The cited portion of *Crow* teaches that if the inode has an empty row, the operating system shifts original extents in the inode by one row. The operating system then inserts a new direct extent into the newly emptied row of the inode. The operating system then writes the new file segment to a new data block. (Applicants note that the file segment is written to the data block to which the extent points, and not to the inode itself).

However, *Crow* in no way teaches that, "the partially filled block is a *last* block of the another file." In fact, *Crow* does not teach partially filled blocks at all, so *Crow* cannot teach this claimed feature. Certainly, *Crow* does not mention that a partially filled block is a *last* block of the another file, as claimed. Accordingly, *Crow* does not anticipate claim 4 or any other claim in this grouping of claims.

I.B.3. Claims 5, 13, and 20

Claim 5 is a representative claim of this grouping of claims. Claim 5 is as follows:

> 5. The method of claim 1, wherein the space is located in an extension area in the inode.

The Examiner rejects claim 4 as anticipated by *Crow*. Regarding claim 5, the Examiner states that:

> Crow discloses in figures 5-10, wherein the space being located in an extension area in the inode.

Office Action dated December 27, 2007, p. 4.

None of Figures 5-10 teach the claimed feature that, "the space is located in an extension area in the inode." The Examiner does not point to any particular part of these figures as teaching this claimed feature. Instead of showing that each individual figure fails to show the features of claim 4, Applicants use Figure 7 of *Crow* to prove that the opposite is true; namely, that *figures in Crow specifically do not teach* that the space is located in an extension area in the inode, as claimed. A similar analysis applies to each of Figures 5, 6, and 8-10. Figure 7 of *Crow* is as follows:

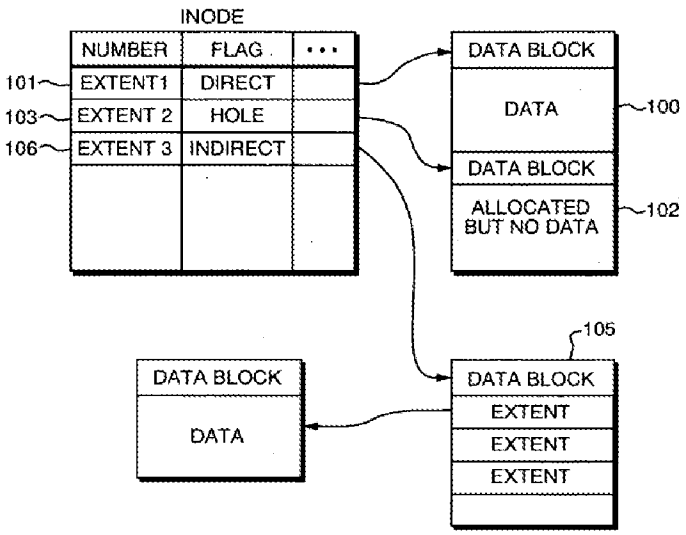

FIG. 7

Figure 7 of *Crow* shows that extents in an inode *point to* data blocks. The *data blocks* contain the stored data. The inode extents do not store the data, only the data blocks store data. Therefore, these figures do not teach the claimed feature of, "wherein the space is located in an *extension area* in the inode." Accordingly, *Crow* does not anticipate claim 5 or any other claim in this grouping of claims.

The portion of *Crow* closest to the features of claim 5 is as follows:

> [0053] The operating system writes the binary value to the third entry 176 to indicate storage of a data file when the associated inode is first created. Then, the operating system uses the inode to store the associated data file. When the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.

Crow, paragraph 53.

This portion of *Crow* states that the operating system uses the inode to store the associated data file. When the size of the data file surpasses the space in the inode, the operating system converts the inode to an inode for storage of lists of extents.

However, again *Crow* does not teach that the data is stored in the *extents themselves*, as required by claim 5. Therefore, this portion of *Crow* does not teach all of the features of claim 5. Moreover, no portion of *Crow* teaches these claimed features. Accordingly, *Crow* does not anticipate claim 5 or any other claim in this grouping of claims.

I.B.4. Claims 6, 14, and 21

Claim 6 is a representative claim of this grouping of claims. Claim 6 is as follows:

> 6. The method of claim 1 further comprising:
> determining whether a file size for the data is divisible by a block size for blocks in the file system; and

if the file size is divisible by the block size, storing the data in a block.

The Examiner rejects claim 6 as anticipated by *Crow*. Regarding claim 6, the Examiner states that:

> *Crow* discloses further comprising determining whether a file size for the data being divisible by a block size for blocks in the file system; and if the file size is divisible by the block size, storing the data in a block (paragraph [0031], [0034]).

Office Action dated December 27, 2007, p. 4.

The first portion of *Crow* cited by the Examiner is as follows:

> [0031] Each data block 80-82, 84-85, 92-94 has the same size, for example, 4K bytes. Nevertheless, the extents 65-66 can map file segments of different sizes to physical storage locations. To handle file segments of different sizes, each extent has a length field that indicates the number of data blocks in the string of data blocks that stores the associated file segment.

Crow, paragraph 0031.

This portion of *Crow* teaches that data blocks have the same size. Extents in an inode can map file segments of different sizes to different physical storage locations. An extent length field in the inode indicates the number of data blocks in the string of data blocks that stores the associated file segment.

However, *Crow* does not teach *determining* whether a file size is divisible by a block size for blocks in the file system. *Crow* does not actually store data in a block ***if*** the file size is divisible by the block size, as claimed. Instead, *Crow* only teaches that large files are stored by appending small data blocks. Because this feature is not equivalent to the claimed feature, this portion of *Crow* does not teach the features of claim 6.

Nevertheless, the Examiner quotes from a second portion of *Crow* as teaching the features of claim 6. The second portion of *Crow* cited by the Examiner is as follows:

> [0034] The address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. In one embodiment, the pointer fields for the logical volume and the data block therein are 2 bytes and 4 bytes long, respectively. For this field size and data blocks of 32 kilobytes, the extent fields can identify about 140.times.10.sup.12 bytes of data in each of about 64K different logical volumes. Thus, the file system of the distributed storage system 40 can handle very large files.

Crow, paragraph 0034.

This portion of *Crow* teaches that the address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. *Crow* also describes how the described system can handle very large files using small data blocks. However, again, *Crow* does not teach *determining* whether a file size is divisible by a block size for blocks in the file system. *Crow* does not actually store data in a block ***if*** the file size is divisible by the block size, as claimed. Instead, *Crow* only teaches that

large files are stored by appending small data blocks among different logical and physical volumes. Because this feature is not equivalent to the claimed feature, this portion of *Crow* does not teach the features of claim 6.

Neither of the above portions nor any other sections of *Crow* teach all of the features of claim 6. Therefore, *Crow* does not anticipate claim 6 or any other claim in this grouping of claims.

II. Conclusion

The subject application is patentable over the cited references and should now be in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: March 27, 2008

Respectfully submitted,

Theodore D. Fay, III/

Theodore D. Fay, III
Reg. No. 48,504
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants

TDF/ljm